NBSIR 88-3787

May 19, 1988

IMPLEMENTATION OF THE EXECUTION CONTROL SYSTEM OF THE INSPECTION WORKSTATION

By:
Howard T. Moncarz

AMRF

U.S. DEPARTMENT OF COMMERCE National Bureau of Standards Gaithersburg, Maryland

IMPLEMENTATION OF THE EXECUTION CONTROL SYSTEM OF THE INSPECTION WORKSTATION

Howard T. Moncarz

May 19, 1988

TABLE OF CONTENTS

LIST OF FIGURES

IMPLEMENTATION OF THE EXECUTION CONTROL SYSTEM OF THE INSPECTION WORKSTATION

I. INTRODUCTION

1. WHAT THIS DOCUMENT IS ABOUT

This document describes the benchmark (March, 1987) implementation of the execution control system (ECS) of the Inspection Workstation (IWS). ECS is a program that runs on all the controllers in the IWS and embodies the design principles of the AMRF, as described in the IWS document Architecture and Principles of the Inspection Workstation [A.1]. The ECS program includes the network module, which supports local communications among the IWS controllers and global communications between the IWS and the AMRF. The general AMRF network software is discussed in the document, AMRF Network Communication [B.1].

The control structure of the IWS consists of four controllers that are networked together. The implementation of these controllers, each of which runs under the control of ECS, are described in the four IWS documents, Implementation of the CMM Controller [A.3], Implementation of the Inspection Robot Controller [A.5], Implementation of the SRI Controller [A.4], and Implementation of the Inspection Workstation Controller [A.6].

2. AUDIENCE

Anyone who needs to understand the internals of the IWS software should read this document. This includes anyone who will continue the development of the IWS or plans to make modifications to it.

3. OVERVIEW

The next chapter, Chapter II, discusses the computers and software environment used in the IWS. Chapter III presents a top level understanding of the ECS program, and Chapter IV goes into the ECS design specifics.

The appendices contain a list of the documents in the IWS series (Appendix A), other references cited (Appendix B), a glossary of terms used in this document (Appendix C), and a reader comment form to solicit feedback.

II. IMPLEMENTATION CONSIDERATIONS

When the IWS project was started, the computers and systems software had already been acquired. This chapter will not attempt to rationalize those purchases, but will briefly describe the hardware and software environment under which the IWS was developed and is now an operational workstation in the AMRF.

Within the broad mandate of the AMRF to utilize, as much as possible, off-the-shelf equipment from a wide variety of manufacturers, the computer hardware and software chosen for the IWS represent a good selection. However, please note the disclaimer printed on the title page of this document.

1. COMPUTERS

The computer system selected for the IWS is the 200 Series of the Hewlett Packard (HP) desktop computers. Four computer systems are used in the IWS, one for each controller. Each system consists minimally of the CPU, monitor, keyboard, hard disk drive, and floppy disk drives. The hard disk is used to store programs and data, and the floppies are used to install systems software and to transfer software between computer systems. Additionally, each computer system contains the I/O boards required to interface that computer to the equipment under its control or to other controllers.

2. SOFTWARE

Pascal and BASIC were the two computer languages that were purchased with the computers. Pascal was chosen for the IWS software development because it is the more structured language of the two. The extensions to the language provided by HP were used to make the software development easier, and in fact, the design generality desired would not be possible with standard Pascal. In particular, separate module compilation was considered essential. Also, heavy use was made of the relaxed type checking available in the extension, as well as heavy use of the systems level procedure libraries provided.

A complete software development environment, similar to UCSD (University of California at San Diego) Pascal, was provided by HP in support of its Pascal and was used for the IWS project. This operating system will be referred to as the HP Pascal operating system in this document.

III. THE EXECUTION CONTROL SYSTEM (ECS)

ECS is a program that runs on each controller computer and incorporates the design principles of the AMRF. These principles utilize hierarchical task decomposition, data-driven control, state machines, common memory, and network communications. The ECS program loads and executes the state machine modules required to make the computer on which it is running operate as a specific controller, be it the Robot, CMM, SRI, or Workstation Controller.

1. THE GENERIC CONTROLLER

When designing a specific controller, the main tasks that it will perform should be specified and then decomposed hierarchically into successively simpler tasks. The tasks performed at each level of decomposition are grouped together into separate modules.

All IWS controllers have a similar decomposition (see Figure 1). The top level module of the generic controller interfaces it to its supervisor controller and implements the UVA model, defining the start up and shut down protocols. The next level module performs the main tasks that the controller is designed to do. This is followed by modules that decompose the top level task into successively simpler ones. The bottom level module interfaces the controller to the actual equipment that it will control or to another controller.

2. STATE MACHINES

The modules that make up the controller described in Section 1 are implemented as state machines (abbreviated FSM). (Technically the IWS uses state machines, not **finite** state machines, but the abbreviation FSM will still be used.)

ECS is itself comprised of separate software modules, each containing functions and/or data structures. These modules should not be confused with the state machine modules that make up the controller program. To avoid confusion, state machine modules will be referred to as FSMs throughout the rest of this document.

The main business of the ECS program is to load and run these FSMs and to provide communications among them. Consequently, it behooves us to describe the FSMs in more detail.

Consider the FSM as a black box as shown in Figure 2. The inputs are derived from other FSMs, from data collected or processed at its own level (sensory data), and from the FSM's localized view of the system (the world model). The black box processes these inputs and produces outputs. Since the black box is a state

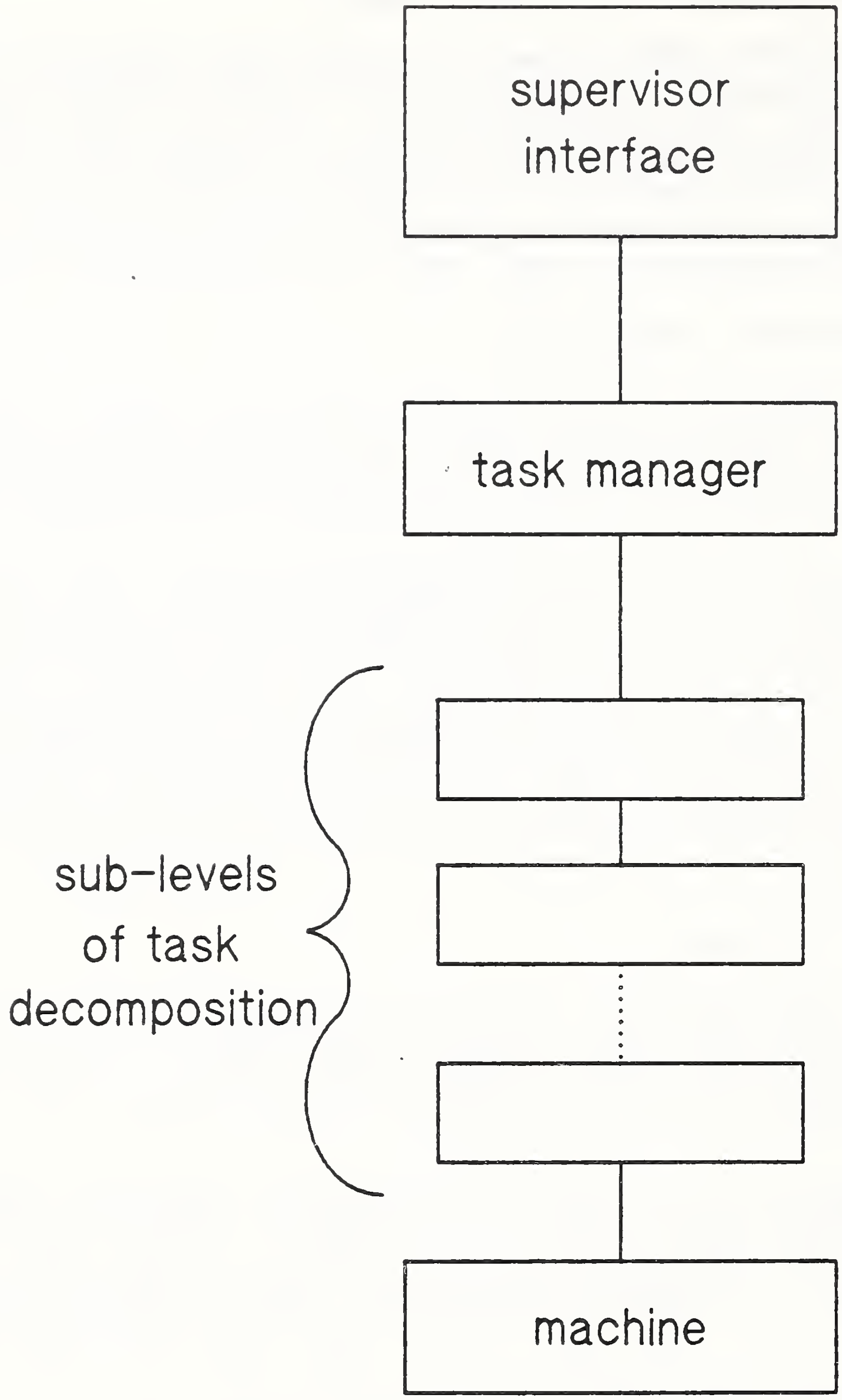

Figure 1. Control levels for the IWS Generic Controller

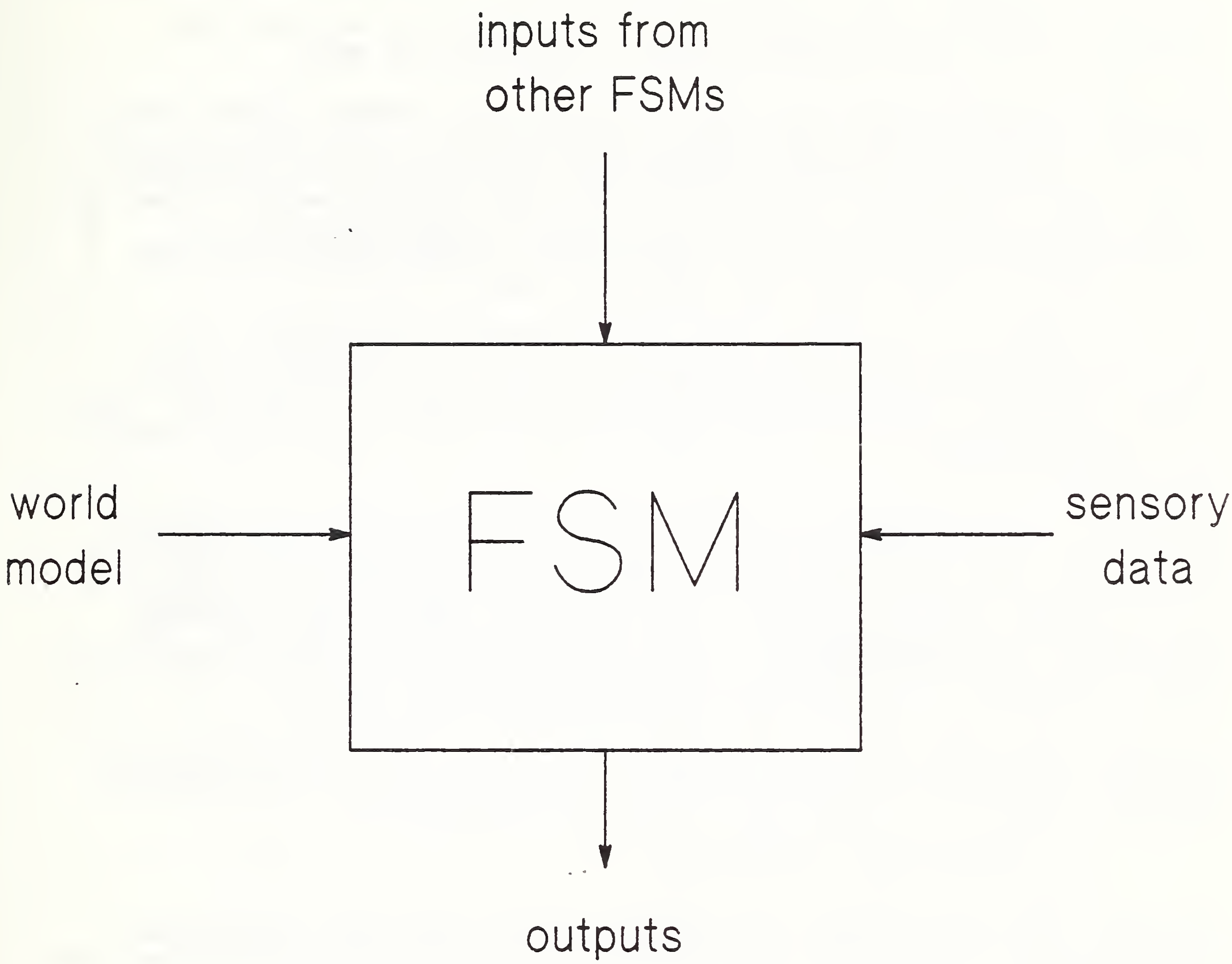

Figure 2. External View of State Machine (FSM)

machine, however, the outputs are dependent not only on the inputs, but on the internal state of the FSM as well.

All FSMs communicate with other FSMs through common memory (CM). Figure 3 shows a very simplified description of CM communication. Certain variables in each FSM can be declared as CM variables. Variables declared as outputs are transferred by the CM system to global CM. Similarly, variables declared as inputs are transferred by the CM system from global common memory to the local memory of the FSM that declared them.

Figure 4 shows the major software components of a single FSM. Each FSM contains two main procedures--one to declare all CM variables, and one to actually perform the logic of the FSM. This latter procedure uses three types of variables--temporary, state, and common memory.

Temporary variables are created as needed each time the FSM is executed. The values of these variables are not needed beyond that time. On the other hand, the values of state variables are saved between FSM executions, and as described above, are what characterize state machines. The third type of variables, CM variables, are transferred to and from global CM, and provide the means for FSMs to communicate with each other.

Finally, each FSM can access data stored in local disk files. A generic interface to these files is provided as part of the ECS program. (This is described further in Section 9.)

3. LOADING THE CONTROLLER

ECS works in two phases. The first phase loads the controller program, and the second phase runs it. This section describes the first phase.

When the ECS program is first started, it reads a data file called the "Load File" (Figure 5). This file designates what software components must be loaded from disk to run a particular controller.

The Load File first specifies which local data files are required and the formats for those files. ECS prepares its internal data structures so that those local data files can be accessed during the execution phase of the controller.

The procedure files are selected next by the Load File, and loaded into RAM by ECS. These files contain procedures that the FSMs, yet to be loaded, will use during the execution phase of ECS.

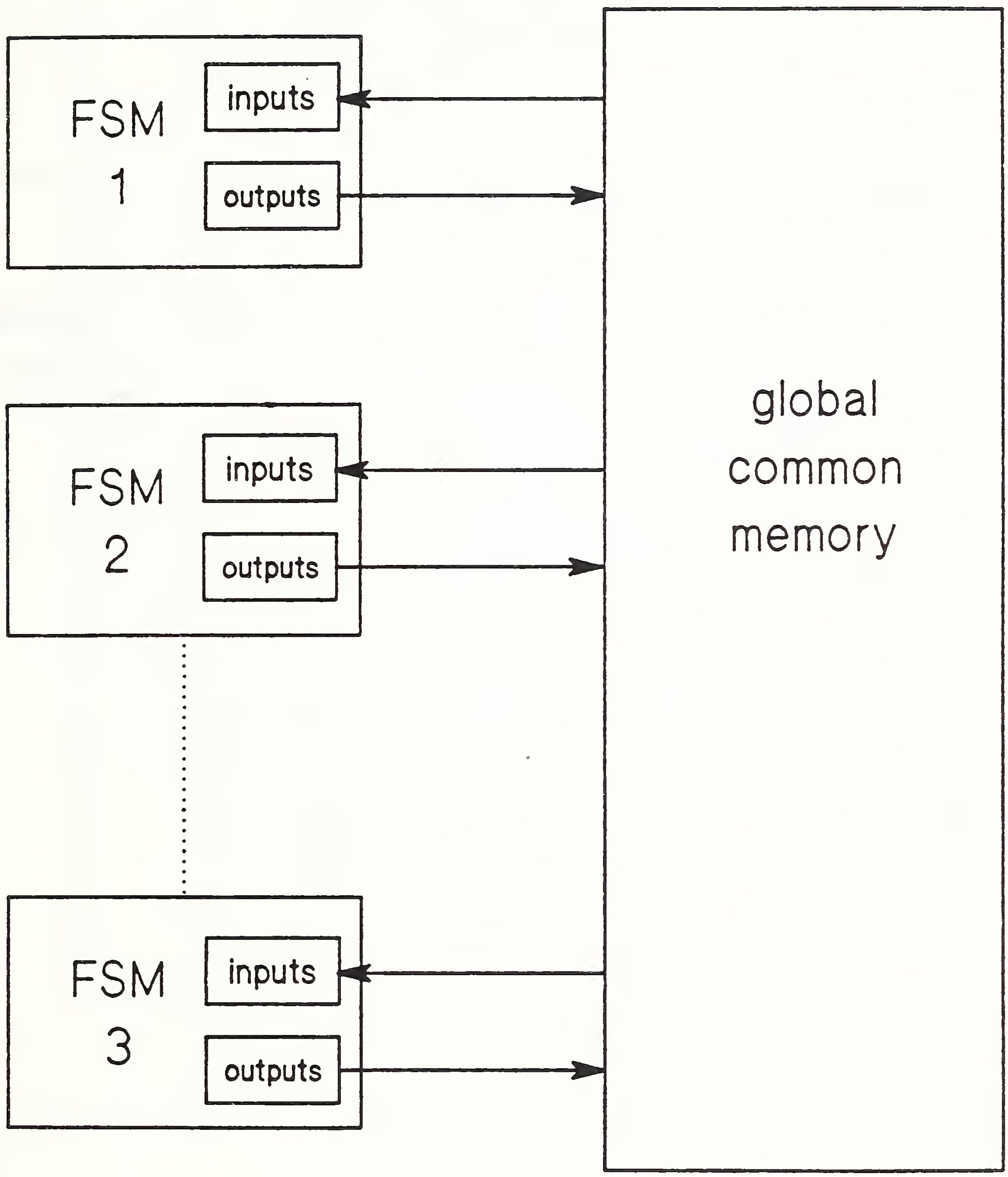

Figure 3. Communication by Common Memory (CM)

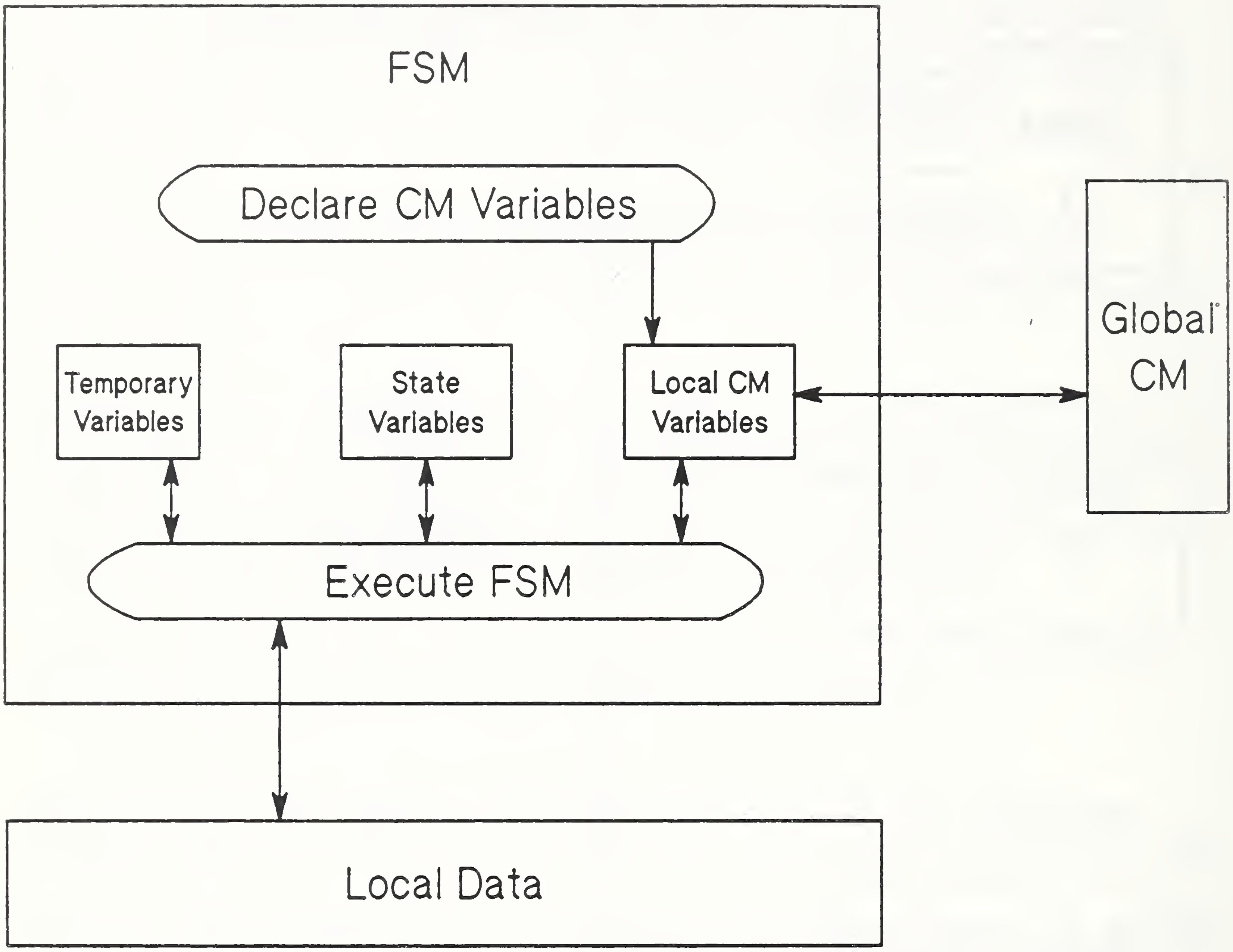

Figure 4. Software Components of FSM

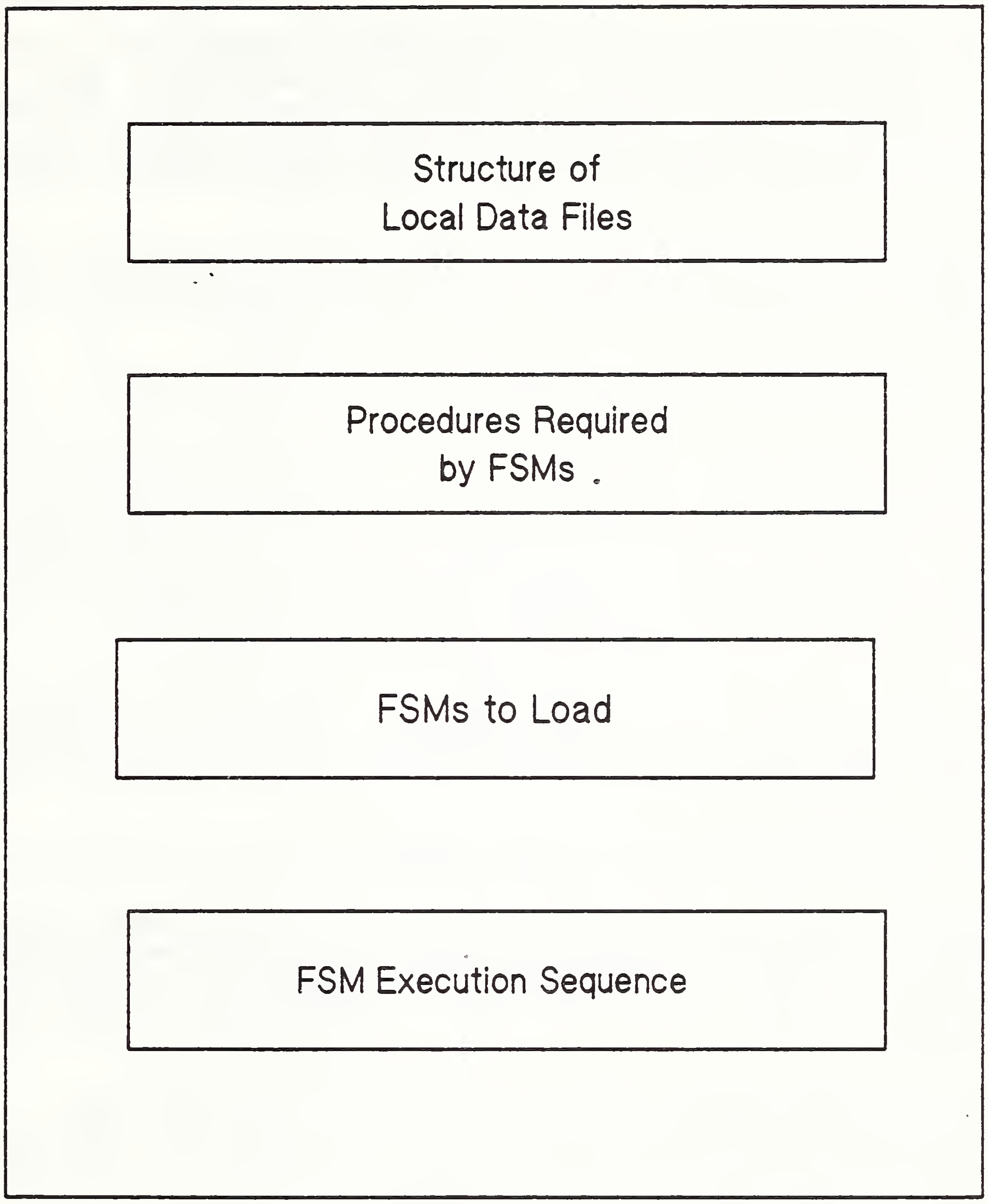

Figure 5. Components of the Load File

The Load File now selects which FSMs must be loaded to run the particular controller desired. As each FSM is loaded, the procedure to declare CM variables is performed. After all FSMs are loaded, and all CM variables consequently declared, the order in which the FSMs will be executed is then read from the Load File.

At this point, the loading phase of ECS is completed, and ECS may now go into its execution phase, running the controller it has just loaded.

4. EXECUTING THE CONTROLLER

In the execution phase, ECS goes into a loop which is executed continuously. One cycle of this loop is shown in Figure 6. Each FSM is run in turn, according to the sequence specified in the Load File as discussed in Section 3. After all FSMs have been run, common memory variables are transferred, and then network variables are transferred. Figure 7 is a simple graphic illustrating the data transfers for one controller.

This completes one ECS cycle, and the next one begins. This continues indefinitely until the operator stops the program by pressing the STOP key at the HP keyboard (unless an error crashes the program prematurely).

5. COMMUNICATING VIA COMMON MEMORY

As already mentioned, FSMs communicate with one another through common memory. Any variables that will be transferred through common memory must be declared when the controller program is loaded. This is done in the "Declare CM Variables" procedure shown in Figure 4.

For any CM variable, only one FSM is allowed to write that variable to common memory, but any number of FSMs may read it. It is the responsibility of the controller implementor to enforce this rule when programming the controller.

CM variables are transferred once every ECS cycle, after the controller FSMs execute. As shown in Figure 7, the procedure to transfer CM variables delivers output variables from the local memory of each FSM to global CM, and vice versa for input variables. Because CM variables are transferred once a cycle, a command sent from one FSM to another will not be received by the latter until the next cycle.

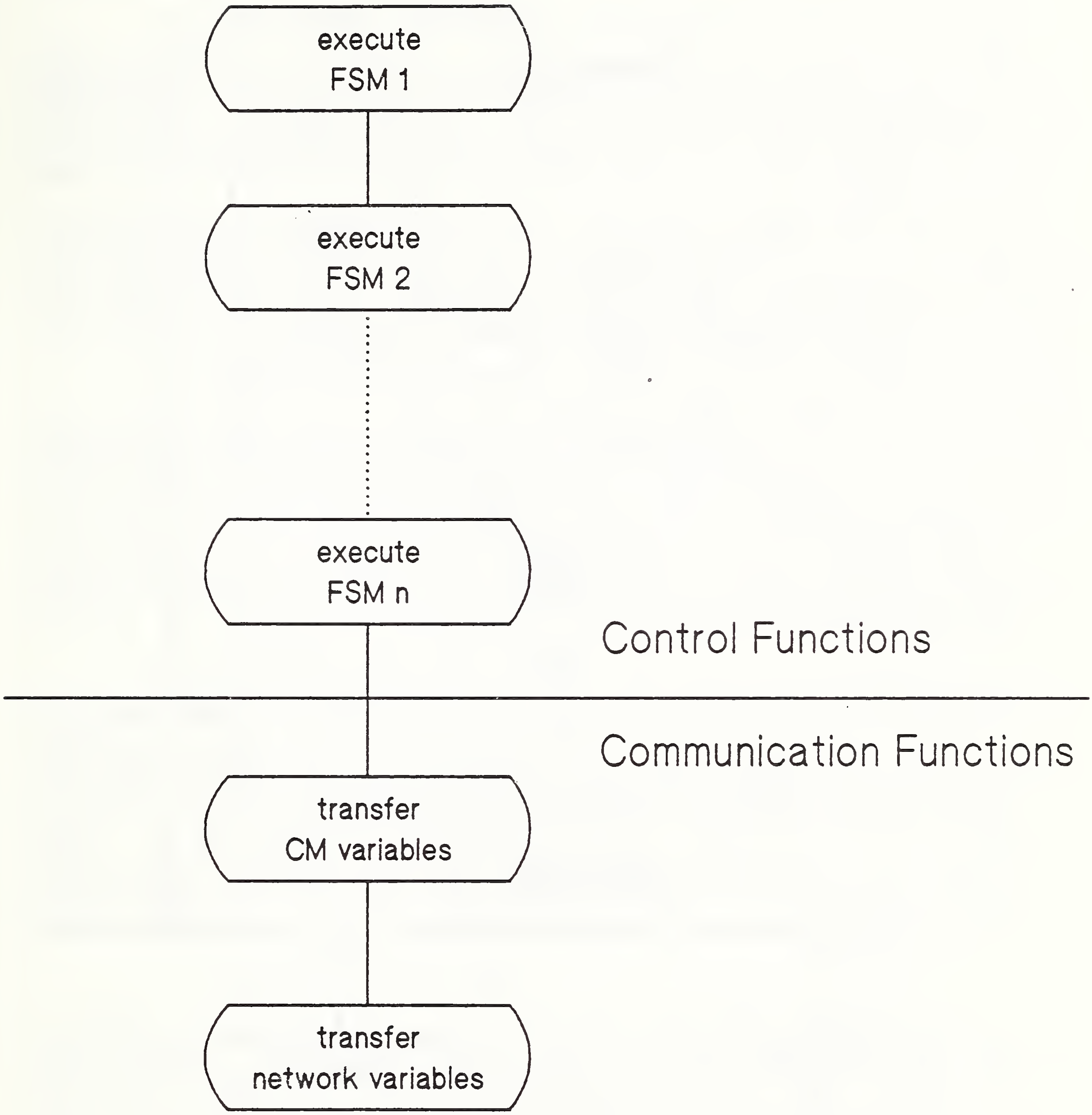

Figure 6. One ECS Cycle

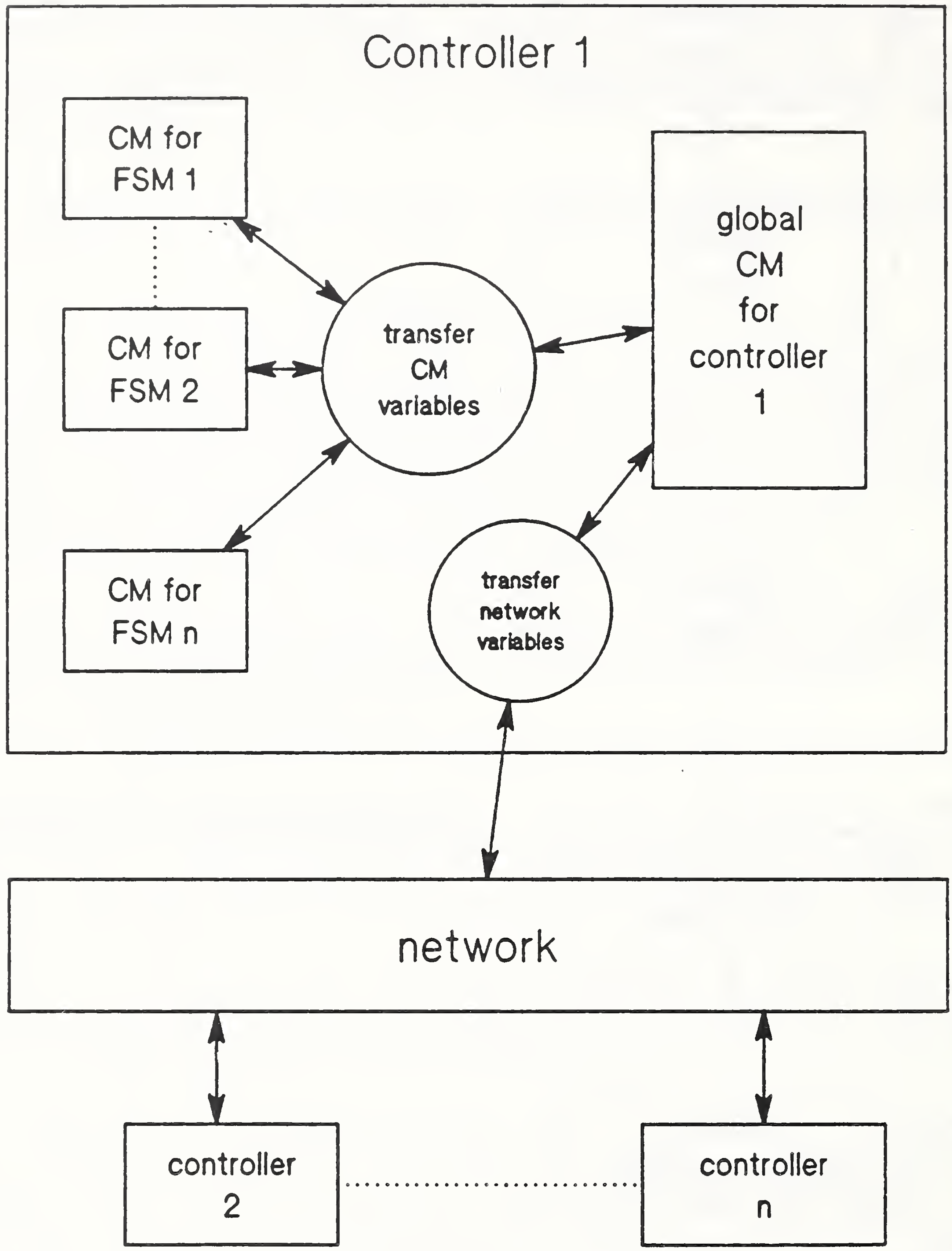

Figure 7. CM and Network Communications

6. THE NETWORK INTERFACE

Referring again to Figure 7, the procedure to "transfer network variables" transfers selected variables from global CM of Controller 1 to the network. Similar procedures on the other controllers connected to the network transfer variables from the network to and from their own global CM.

The particular variables selected for each controller are specified by a script file that is read when ECS is first started. Once the network is established, network variables are transferred once every ECS cycle. Details of the network implementation are described in the document, AMRF Network Communication [B.1].

7. GENERAL ERROR RECOVERY

Most errors that may occur during controller operation are specific to the controller executing, and the error handling must be included in that controller's FSMs. However, the module "errors" has been included as part of the ECS program. It may be accessed by the controller program to either exit the program gracefully, or to continue the ECS program from a known state after the controller error has been handled successfully. Further details for the errors module are presented in the next chapter.

8. ACCESSING THE AMRF IMDAS

The AMRF IMDAS (Integrated Manufacturing Data Administration System) is not accessed directly by the ECS program. (IMDAS is the distributed data system which provides common interfaces to the AMRF's user programs and underlying databases [B.2, B.3].) Rather, a separate FSM (the data server) is used to handle this task. In the current IWS implementation, only the Workstation Controller accesses the AMRF IMDAS, and consequently the data server implementation is described in the IWS document, Implementation of the Inspection Workstation Controller [A.2].

9. ACCESSING LOCAL DATA

As shown in Figure 2, one of the data inputs to an FSM is the world model. Depending on the level of task decomposition that a particular FSM represents, the world model for that FSM should be at a similar level of data abstraction. For example, the top level task in the CMM Controller needs to know what part is to be inspected and what inspection plan to use, and that information constitutes the world model for that FSM. However, at an FSM several levels lower, the world model consists of the specific points on the part to inspect.

In the current implementation, the data required is contained in data files already residing on the local controller computer system. All of this "world model" data should come from the AMRF IMDAS, and a future upgrade to the IWS software will provide this. However, since IMDAS is not dedicated to the IWS, the task level FSM should retrieve all the data required by the controller in one piece and transfer that data to these local data files that can be accessed by any FSM as required in real-time.

The interface between any controller and the local data files is composed of two parts. The first part is a set of procedures that specifies the particular data required by the controller, and is consequently a part of that controller (and contained in a module called "get_data"). The second part is a data independent set of procedures used to search and retrieve the data required, using procedures from get_data as needed. This latter part is contained in the module called "flatfile" of the ECS program, and is explained in the next chapter.

IV. DESIGN SPECIFICS

This chapter describes the modules that comprise the ECS program. Each module contains a set of functions and/or data structures used in ECS. Some of these modules are grouped together and collectively referred to as a library module.

1. HP DIRECTORY STRUCTURE

The HP Pascal operating system allows multiple directories to be created on the hard disk drives. Each directory can contain multiple files, but the directories cannot be nested.

This directory system was used for configuration control. All directories created that are associated with ECS or any of the IWS controllers have names composed of two parts. The first part is a three letter abbreviation for the directory's main name. The second part contains the underscore character followed by a number representing the version number of the software. For the benchmark software, which is documented here, that version number is 3.

The main directory names for the four IWS controllers are wsc, cmm, irc, and sri representing the Workstation Controller, the Coordinate Measuring Machine Controller, the Inspection Robot Controller, and the Surface Roughness Instrument Controller, respectively. The ECS program and its constituent parts reside on the directory named env. Env stands for the environment, since each controller can be considered to run in an environment consisting of the ECS program, and references other modules that are best stored in this directory. The network software resides in another directory called "net".

The normal procedure in running a controller program is to go to the controller directory ("prefix" to the controller directory in HP terminology). The ECS program is then started from the controller directory by executing the command env_3:ecs, to execute the benchmark version of the ECS program.

2. DESCRIPTION OF ECS MODULES

The modules comprising ECS are ecs_main, ecs_init, st_lib, mail_mgr, flatfile, errors, cmd_stat, universal, and net_lib. Of these, st_lib, mail_mgr, universal, and net_lib are library modules.

2.1. ecs_main and ecs_init

Ecs_main is the main supervisor for the ECS program. It directs the initialization of the system, the loading of the controller, the initialization of the network, and finally, the execution of the FSMs.

Ecs_init executes the procedure, init_debug_options, and then calls the procedure from the mail_mgr module to initialize the common memory system.

Init_debug_options allows the user to set the debug options before loading and running the controller. This procedure displays a menu of options that are read from a data file named D_MENU.TEXT, which is stored in the directory of the controller being executed. Thus, a different D_MENU.TEXT file may be stored in each of the controller directories.

Options are selected by entering individual characters in a string. A user can select various options and run the program under different conditions, mainly for debugging purposes.

2.2. st_lib

St_lib is a library module that contains the three modules: loadfile, st_load, and st_xqt.

The loadfile module contains procedures to allow ECS to access the Load File. (Note: do not confuse the "loadfile" program module with the "Load File" data file.) St_load uses these procedures to read the Load File and load up the controller components from disk (see Chapter III, Section 3). As each FSM is loaded into memory, its procedure to declare CM variables is located and executed. This establishes the CM variables for each FSM.

The last thing read from the Load File is the order in which the FSMs should be executed. As the procedure to execute each FSM is located, a record containing the starting address for that FSM is created and added to the single linked list of FSMs in the order they should be executed.

Finally, st_xqt executes the controller (see Chapter III, Section 4). The linked list described above is scanned from its beginning, and each FSM pointed to by it is executed. After all FSMs in the list are executed, a procedure is called from the mail_mgr module to transfer all CM variables. The network variables are transferred at some time during this cycle, other than the time that the CM variables are being transferred. The

exact time that those network variables are transferred depends on the net_lib module and is done in background mode (under interrupt control).

2.3. mail_mgr

The mail_mgr library module is responsible for handling common memory. This module itself contains six modules--sllprims, cms_data, alloc, cmsprims, allo_cms, and xqt_xfers.

The mail_mgr creates and manages three single linked lists of records. The first list is the list of CM variables. Each record in the list contains the name of the variable, its size in bytes, and its address in the global common memory. The other two lists specify the CM variable transfers from global common memory to an FSM's local memory, and vice versa. A transfer from local memory to the global common memory is called an output data transfer, and the reverse transfer is called an input data transfer. All output data transfers are contained in the list of output data transfers, and all input data transfers are contained in the list of input data transfers. The information contained in each record, for either of these two lists, is the size of the local variable, its address in local FSM memory, and a pointer to the record in the CM variable list, which in turn references the variable in global common memory. The data structures described here are contained in the module cms_data.

The functions contained in the module sllprims manage a general single linked list (sll). Each record in this list has two fields--the first field contains a pointer (data_ptr) to a general data structure, and the second points to the next record in the list. This approach permits the same functions to handle single linked lists containing different types of data. The functions include "addl" to add a node to the sll, "find" to locate a particular record in the sll, "top" to go to the first record, and "next_node" to move to the next record.

The module cmsprims contains functions used by the sllprims functions to operate on the particular data structure pointed to by data_ptr. These functions include creating a record of the correct data structure type as well as matching a particular field in that data structure--the latter used by the sllprims find function.

The alloc module manages bytes. Initially it reserves a fixed block of memory to be used for the global common memory. Whenever space is required to create a new CM variable, the function, allocate, secures that memory and returns a pointer to where it is located.

The only two modules that are directly accessed by ECS are xqt_xfers and allo_cms. Alloc_cms contains the procedure cm_declare. The FSM declare procedure contains a cm_declare procedure for each CM variable declared by that FSM. Cm_declare checks the CM variable list to see if this variable has been allocated in global common memory. If it hasn't been, then space for it is allocated, and a record is added to the CM variable list which references it. Next, a record is added to the input or output data transfer list (depending on whether the variable has been declared as an input or output) that references that local variable.

The module xqt_xfers contains the procedure xqt_data_transfers. This procedure is called by st_xqt (in the st_lib module) and handles the common memory data transfers. First the output data transfers list is scanned. All variables referenced in this list are copied from their local memory location to global common memory. Next the input data transfers list is scanned, and all variables in this list are copied from global common memory to the proper local variable locations.

2.4. flatfile

The flatfile module contains four procedures that are used by ECS and the controller program to access the local data files. These procedures are initialize_database, name_flatfiles, first_data, and next_data.

ECS calls initialize_database after it reads the Load File. The Load File specifies the formats for the flat files used by the particular controller being executed, and initialize_database initializes the ECS program with that structure.

The names for all local data files are composed of two parts. The first part names the particular flat file (the relation). A different name is used for each relation and is specified by the Load File. The second part, called the extension, consists of the underscore character and three letters. The extension is the same for all of the flat files, and names a particular set of local data files. The extension is specified by the procedure name_flatfiles, and is called from the controller program. The extension, and consequently the set of local data files, may be changed during the controller execution by calling this procedure each time a change is required.

All the flat files are composed of ASCII characters that are broken up into records, each record containing one or more fields. Records are separated by a carriage return and a line feed. Fields are separated by one or more spaces.

There are two types of fields--key fields and data fields. Key fields are used to find the particular record in the data file that is required. The data fields contain the data required. The Load File specifies which fields are key fields and which are data fields for each data file (relation).

The two functions, first_data and next_data, are used to retrieve the actual data. First_data locates the data record using the key fields, stores the data fields in a results array, and returns the first of these "results". Thereafter, each time next_data is called, that function returns the next result string from the results array. (Note that it is the responsibility of the procedures contained in the get_data module in the controller program to convert these result strings to the data types required and to match them to the proper variable names.)

2.5. errors

The two procedures in the errors module, exit_program and handle_error, implement the general error recovery discussed in Chapter III, Section 7. The procedure, exit_program, allows the controller implementor to exit the ECS program gracefully, either by pressing the stop key or by sending a software stop. Handle_error is a procedure that allows the implementor the option of continuing the ECS program after an error, or exiting it by calling the previous procedure, exit_program. In the former case, the operator has the option of displaying an error message, waiting for the user to press the ENTER key before continuing, and to pop to a higher level of HP error handling.

2.6. cmd_stat

A task module in a controller communicates with its subordinate by sending it a command and receiving a status report back from that subordinate on the progress of executing that command. The data structures and functions used in this communication are contained in the module cmd_stat.

2.7. universal

Universal is a library module which contains the modules globdefs and funcs. The module, globdefs, includes data structure types that are used throughout the ECS program. It also includes the string variable "debug" which is used to set the debug options (in procedure, init_debug_options, in module, ecs_init) for each run. The module, funcs, contains low level general functions that are used throughout ECS. The data structures and functions in these two modules are also widely used in the controller programs that are run by ECS.

2.8. net_lib

Net_lib is a large library of modules that are used to operate the IWS network and interface it to the ECS program. In large part, the network software is not documented here, nor in any of the IWS documentation. For a comprehensive treatment of the AMRF network software, refer to the AMRF document, <u>AMRF Network Communications</u> [B.1].

The procedures and functions from net_lib that are directly referenced from ECS are nipini, copy_done, disable_nip, enable_nip, and clkclose. The procedure nipini initializes the network software. It is also responsible for communicating with the network software on the other IWS controllers and the AMRF Cell Controller to establish the network, based on a script file located on the Workstation Controller.

The function, copy_done, is false whenever the network is transferring bytes from the controller. The common memory transfers are not allowed to take place at the same time that the network is transferring its variables. To prevent this, the procedure, disable_nip, is called to disable network data transfers. The procedure, enable_nip, is called after the CM data transfers to reenable the network data transfers. Finally, just before ECS is stopped, the procedure, clkclose, is called to disable the network interrupt clock.

3. LINKING IT ALL TOGETHER

The separate modules comprising each library module described in Section 2 are linked together. Then, all of these together with the other modules are linked to build the ECS program, contained in the ecs.CODE file. The HP linker is used to accomplish this. For ease of accomplishing this, the HP stream command is used.

To run a particular controller, the user should "prefix" to that controller directory. Then the ECS program should be run from that directory. The D_MENU file will be displayed, and the user will be prompted to enter the debug string for this run. Next, the user must enter the Load File which specifies which controller to run and its configuration.

APPENDICES

A. IWS DOCUMENTATION LIST

1. H. T. Moncarz, Architecture and Principles of the Inspection Workstation, to be published as an NBSIR, 1988.

2. H. T. Moncarz, Implementation of the Execution Control System of the Inspection Workstation, NBSIR 88-3787, May 19, 1988.

3. H. T. Moncarz and T. H. Hopp, Implementation of the CMM Controller, to be published as an NBSIR, 1988.

4. H. T. Moncarz and T. V. Vorburger, Implementation of the SRI Controller, to be published as an NBSIR, 1988.

5. H. T. Moncarz and B. Borchardt, Implementation of the Inspection Robot Controller, NBSIR 88-3772, April 21, 1988.

6. S. A. Osella, Implementation of the Workstation Controller, to be published as an NBSIR, 1988.

7. J. Zimmerman, Inventory of Equipment in the Inspection Workstation, to be published as an NBSIR, 1988.

8. H. T. Moncarz, S. A. Osella, B. Borchardt, and R. Veale, Operations Manual for the Inspection Workstation, NBSIR 88-3766, April 21, 1988.

9. J. Zimmerman, Recommended Technical Specifications for Procurement of Commercially Available Systems for the Inspection Workstation, NBSIR 88-3779, 1988.

B. REFERENCES

1. R. Rybczynski, et al, AMRF Network Communications, to be published as an NBSIR, 1988.

2. D. Libes and E. Barkmeyer, "The integrated manufacturing data administration system (IMDAS)--an overview", International Journal of Computer Integrated Manufacturing, Vol. 1, No. 1, pp. 44-49.

3. C. Furlani, et al, "The Integrated Manufacturing Data Administration System (IMDAS)", to be published as an NBSIR, 1988.

C. GLOSSARY (and abbreviations)

benchmark software
: Status of the IWS software at the benchmark date of March 26, 1987.

CM
: Abbreviation for common memory

CMM
: Abbreviation for the Coordinate Measuring Machine.

common memory system
: Manages communications between state machines.

controller
: Supervises the operation of a mechanism, another controller, or both.

coordinate measuring machine
: Machine used to measure the dimensions of a part.

data server
: Software module that interfaces the controller it resides on to the data it requires.

ECS
: Abbreviation for the execution control system.

execution control system
: Computer program that runs on each controller computer and implements the AMRF design principles. This program loads and executes those modules that determine which controller is actually being run.

FSM
: Abbreviation for finite state machine. Strictly speaking, the software control modules used in the IWS are state machines, not finite state machines. However, for convenience, the abbreviation FSM is kept.

inspection workstation
: AMRF workstation that inspects parts for dimensional tolerance and surface finish.

IWS
: Abbreviation for the Inspection Workstation.

Load File
: Data file that specifies what state machines ECS should load and execute.

network
: The communication system that connects the IWS controllers together, and connects this local network to the AMRF network.

network variable
: Common memory variable that is transferred over the network system from one controller to another.

single linked list (sll)
: List of records in which each record has a pointer to the next record in the list.

SRI
: Abbreviation for the Surface Roughness Instrument.

state machine
: Software control unit that produces outputs that are dependent on its inputs and its internal state. This is the building block for the IWS control software.

state variable
: FSM variable that retains its value between executions of that FSM. (This does not include common memory variables.)

surface roughness instrument
: Machine that measures the optical scattering off the surface of a part that can be correlated with its surface roughness.

UVA Protocol
: Model, proposed by research group from the University of Virginia and adopted by the AMRF, that specifies the start up and shut down sequence for the AMRF as a whole as well as every controller within the AMRF.

WSC
: Abbreviation for the Workstation Controller.

READER COMMENT FORM

IMPLEMENTATION OF THE EXECUTION CONTROL SYSTEM OF THE INSPECTION WORKSTATION

This document is one in a series of publications which document research done at the National Bureau of Standards' Automated Manufacturing Research Facility from 1981 through March, 1987.

You may use this form to comment on the technical content or organization of this document or to contribute suggested editorial changes.

Comments:

If you wish a reply, give your name, company, and complete mailing address:

What is your occupation?

NOTE: This form may not be used to order additional copies of this document or other documents in the series. Copies of AMRF documents are available from NTIS.

Please mail your comments to:

AMRF Program Manager
National Bureau of Standards
Building 220, Room B-111
Gaithersburg, MD 20899

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 88-3787	2. Performing Organ. Report No.	3. Publication Date MAY 1988

4. TITLE AND SUBTITLE

"Implementation of the Execution Control System of the Inspection Workstation"

5. AUTHOR(S)

Howard T. Moncarz

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

This document describes the implementation of the execution control system (ECS) of the Inspection Workstation. ECS is a program that runs on each controller computer and incorporates the design principles of the AMRF. This program loads up and executes the state machine modules required to make the computer on which it is running operate as a specific controller -- be it the robot, coordinate measuring machine, surface roughness instrument, or workstation controller.

The ECS program sits on top of the computer's operating system. In addition to loading and running the proper modules, it provides communications between modules, network communications to other controllers, and a common interface to data.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

AMRF; ECS; execution control system; IWS; Inspection Workstation

13. AVAILABILITY

- [X] Unlimited
- [ ] For Official Distribution. Do Not Release to NTIS
- [ ] Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
- [X] Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

35

15. Price

$11.95

USCOMM-DC 6043-P80